Solar: An Open Platform for Context-Aware Mobile Applications

Guanling Chen and David Kotz

Dept. of Computer Science, Dartmouth College
Hanover, NH, USA 03755
{glchen, dfk}@cs.dartmouth.edu

Abstract

Emerging pervasive computing technologies transform the way we live and work by embedding computation in our surrounding environment. To avoid increasing complexity, and allow the user to concentrate on her tasks, applications in a pervasive computing environment must automatically adapt to their changing *context*, including the user state and the physical and computational environment in which they run. Solar is a middleware platform to help these "context-aware" applications aggregate desired context from heterogeneous sources and to locate environmental services depending on the current context. By moving most of the context computation into the infrastructure, Solar allows applications to run on thin mobile clients more effectively. By providing an open framework to enable dynamic injection of context processing modules, Solar shares these modules across many applications, reducing application development cost and network traffic. By distributing these modules across network nodes and reconfiguring the distribution at runtime, Solar achieves parallelism and online load balancing.

1 Introduction

In a pervasive-computing environment, it is unreasonable to expect a user to configure and manage hundreds of computationally enhanced appliances, particularly when the set of devices and their interactions change as she moves about in the environment. To reduce user distraction, pervasive-computing applications must be aware of the context in which they run. These *context-aware* applications should be able to learn and dynamically adjust their behaviors to the current context, that is, the current state of the user, the current computational environment, and the current physical environment [15], so that the user can focus on her current activity.

Context information is derived from an array of diverse information sources, such as location sensors, weather or traffic sensors, computer-network monitors, and the status of computational or human services. While the raw sensor data may be sufficient for some applications, many require the raw data to be transformed or fused with other sensor data before it is useful. By aggregating many sensor inputs to derive higher-level context, applications can adapt more accurately.

A fundamental challenge in pervasive computing, then, is to *collect* raw data from thousands of diverse sensors, *process* the data into context information, and *disseminate* the information to hundreds of diverse applications running on thousands of devices, while *scaling* to large numbers of sources, applications, and users, *securing* context information from unauthorized uses, and respecting individuals' *privacy*. In this paper we address this fundamental challenge by proposing Solar as an open platform to support context-information collection, aggregation, and dissemination. Its security and privacy features are addressed in another paper [14].

Solar is currently a work in progress. We have implemented a prototype based on Java, with early results and

We gratefully acknowledge the support of the Cisco Systems University Research Program, Microsoft Research, the USENIX Scholars Program, DARPA contract F30602-98-2-0107, and DoD MURI contract F49620-97-1-03821.

In Proceedings of the First International Conference on Pervasive Computing, pages 41-47, June 2002. In an informal companion volume of short papers.

some applications described in a recent report [3]. Solar is now evolving, based on our discussion of the challenges and design guidelines for a context aggregation infrastructure [2]. In this paper, we give an overview of Solar (Section 2), its "operator graph" abstraction (Section 3), and the overall system architecture (Section 4).

2 Solar model

Context-aware applications respond to context changes by adapting to the new context. These applications are active in nature and their actions are triggered by asynchronous occurrences. Thus they are likely to have an "event-driven" structure, where context changes are represented as *events*. We treat sensors of contextual data as *information sources*, whether they sense physical properties such as location, or computational properties such as network bandwidth. An information source *publishes* events indicating its current state or changes to its state. The sequence of events produced are an *event stream*. (A sensor with only a query interface can be easily wrapped with a proxy publisher.) Context-sensitive applications *subscribe* to event streams that interest them, and react to arriving events to adapt to their changing environment.

Solar represents contextual events as a list of hierarchical attribute-value pairs. The internal data structure is a forest with the values at the leaves. An example of an event about the current location of badge numbered "VER16481" may look like Figure 1(a).

To enable an application to subscribe to its desired sources, Solar needs to name the sources and provide a flexible mechanism for resource discovery (services are also named but we focus on sources in this paper). One possibility is to use a hierarchical naming scheme, such as Unix path names. For example, the source that publishes the location of badge "VER16481" could be named [/locator/Versus/VER16481]. The strict hierarchical structure, however, depends on a strong convention that must be followed by both providers and users. The syntax may be difficult to adapt to more expressive name queries (like range selection). These limitations make a hierarchical name space less attractive in a pervasive computing environment.

On the other hand, an attribute-based naming scheme is more flexible and expressive because the order of the attribute-value pairs makes no difference. Solar uses a hybrid approach that keeps the values order-free but allows a tree structure on attribute names (not values) for convenience, exactly like the representation of Solar events. An example name of the source that tracks badge "VER16481" is shown in Figure 1(b).

```
[ badge = "VER16481",
  location =
    [ organization = "Dartmouth",
      building = "Sudikoff",
      room = "120"
    ]
  timestamp = 1023510520
]
```

(a) A location event

```
[ measure = "location",
  badge = "VER16481",
  granularity = "room",
  rate = "3s",
  provider = "Versus"
]
```

(b) A source name

```
$badge-loc = any |
        [ measure = "location", badge = "VER16481" ]

[ device = "camera",
  color = true,
  resolution = "640x480",
  location = $badge-loc:location
]
```

(c) A context-sensitive name for the mobile camera

Figure 1: The event representation and naming mechanism in Solar.

Traditionally a resource directory is fairly static and assumes that names rarely change after they are registered. Context-aware applications, however, may need to look up names based on context. For example, a context-aware display may want to find all nearby cameras. In this case, the physical location is part of the resource description for each camera, which may move frequently. Manual updates to the camera's name are almost impossible in such scenarios. Instead, automatic name updates should be used; for example, attach an active badge to the camera and arrange to have location changes update the camera's name. The name is itself context-sensitive.

Thus there are three challenges involving naming and resource discovery for context-aware applications: to automatically track associations (such as between badge and camera), to handle frequent name updates, and to support persistent name queries so applications can be notified when the name space changes.

Solar uses an approach that allows the name for a source to change according to context. Solar provides

a unique way to automatically manage context-sensitive names by defining the values of contextual attributes to be the output of some sources computing that piece of context. We show an example of such a context-sensitive name for a mobile camera in Figure 1(c). It first defines a source that tracks the camera's location (assuming the camera has badge "VER16481" attached), and then defines the location attribute of that camera to be part of the location event published by the specified source.

Due to space limitations, we reserve the details of Solar's naming system for a future paper.

3 Operator graph

Solar is not simply another event delivery system. Instead, Solar is an open platform to allow dynamic injection of context processing modules that can be shared across applications. The observation is that few context-aware applications want to work directly with raw data from contextual sources. It could be that the application only needs a portion of the data, the data is in wrong format, the data is inaccurate, or the data is incomplete and not useful without aggregating other sensor inputs. Thus, sensor data typically needs to go through several processing steps before it becomes the meaningful contextual knowledge desired by applications. Such contextual computation tends to result in a high development cost for context-aware applications, given the heterogeneous sources and diverse context needed.

On the other hand, we see that many adaptive applications ask for similar (if not exactly the same) contextual information, from basics (such as location context) to high-level social context (such as user activity). It is then natural to re-use the overlapping context aggregation functions or sub-functions among applications. Our approach is to decompose the context-aggregation process of every application into a series of modular and re-usable *operators*, each of which is an object that subscribes to and processes one or more input event streams and publishes another event stream.

Typical operators include filters, transformers, and more complicated aggregators. Some operators do not have accumulated state, such as a transformer mapping from sensor ID to room number. Other operators may have state, such as a temperature aggregator publishing the highest temperature of the day.

From a subscriber's view, both sources and operators expose same interface so they are both event *publishers*. Any publisher can be named, and it is equivalent to say that their output event stream is named since each publisher publishes one event stream, and each event stream has only one publisher.

Since the inputs and output of an operator are all event streams, the applications can use a tree of recursively connected operators (starting from sources) to collect and aggregate desired context. While each application can build its own operator tree, to scale to a large number of applications we must take advantage of opportunities to reuse operators between applications' operator trees. As an open platform, Solar uses a small flexible language to allow applications to specify an operator tree that is to be instantiated at runtime. At the leaves are the name queries for publishers; but some names may match multiple publishers (the application can choose to use any of them or merge all of them into one event stream), and some name queries are resolved to different publishers at different times, depending on context.

Applications can also choose to name the event stream published by the root operator of the subscription tree, so it can be re-used by other applications. These interconnected overlapping operator trees form a directed acyclic graph, which we call the *operator graph*. Currently, Solar incrementally builds the operator graph as new subscription trees are added, sharing event streams wherever names match. We explore the background and justification for this design in an earlier paper [2].

Stateful operators may cause some complexity if shared across many applications. Consider a location aggregator that maintains the current location of all objects that are being tracked, and it publishes an event whenever an object changes its location. An active map now subscribes to this aggregator, but may never know the location of a printer until it moves, although the aggregator has this information in its state. Solar allows the operator to publish a special sequence of events, to the new subscriber only, events that are marked as "state-pushing events" and when considered together represent the current state of the operator. (This feature is reminiscent of the Gryphon expansion operation [1]). Thus, new subscribers receive a series of events to bring them "up to date" and then the ongoing stream of events that represent changes to the current

state.

Occasionally an application may not need the ongoing event stream, but simply needs to obtain the current value. In another system, the application might query the information source. In the operator graph we retain the publish-and-subscribe abstraction by permitting "one-time" subscriptions of stateful operators. An application that needs to obtain the current value of the information published by an operator makes a one-time subscription to that operator. The operator "pushes" its state, as described above, and then cancels the subscription. The one-time subscription approach avoids the need for additional interfaces and maintains the unidirectional data flow.

There are several advantages of the operator-graph abstraction for context collection, aggregation, and dissemination. First, applications receive events semantically closer to their needs than those produced by the sources. Second, due to the modular, object-oriented design we benefit from operator re-usability, data abstraction, and maintainability. Third, due to the modular design this operator graph can be deployed across a network and achieve the benefits of parallelism and distribution. Fourth, since filters and aggregators can dramatically reduce traffic along the graph edges, they reduce inter-process (and often inter-host) communication requirements. Finally, by sharing the common operators and event streams the system can support more such applications and more users.

4 System architecture

In this section we describe the various components of the Solar architecture, our current prototype implementation and programming model, and future research directions.

4.1 Overview

The Solar system consists of several components (see Figure 2). A centralized *Star* processes subscription requests from applications and deploys operators onto appropriate *Planets* as necessary. A Planet is an execution platform for Solar sources and operators, and it is responsible for tracking subscriptions and delivering events in the operator graph.

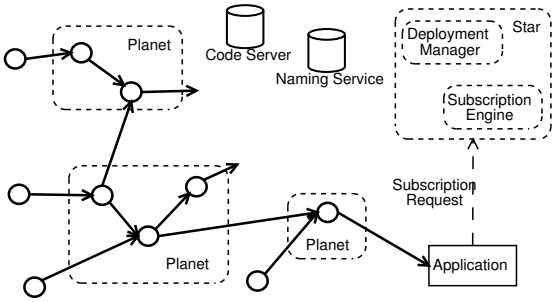

Figure 2: The architecture of Solar. The small circles are sources and operators.

The Star services requests for new subscriptions. When the Star receives a new subscription-tree description, it parses the description, and resolves the name queries on the leaves of the subscription tree using the naming service. It then deploys other operators in the tree by instantiating the operator's object on one of many *Planets*, which periodically register themselves with the Star. Thus the Star maintains a list of active Planets and determines which Planet should host the new operator by considering the Planet's load and network traffic between Planets. In essence, it attempts to map the operator graph onto the Planetary network to distribute load and avoid congestion.

Planets play a key role in the subscriptions of resident operators. When deploying new subscriptions, the Star tells the Planets to arrange a subscription from one of its operators to another operator, possibly in another Planet. Thus the Planet maintains all the subscriptions for each of the resident operators. When an operator publishes an event, the hosting Planet delivers the event to all the subscribing operators (that may reside on several Planets) and applications. When a Planet receives an event, it dispatches the event to the appropriate resident operator(s).

Sources and applications run outside the Solar system and use a small Solar library to interface with Solar. The small library allows the sources to publish events into Solar, and allows the applications to send requests to the Star, to manage their subscriptions, and to receive Solar events over standard network protocols.

4.2 Implementation

Our Solar system is implemented in Java. The first prototype models events as arbitrary Java objects and uses Java serialization for event transmission [3]. In our second prototype we use the hybrid hierarchical attribute-value structure to represent events (see Section 2) and are enhancing event delivery performance [19]. The operators are small Java objects that implement a simple publish/subscribe interface.

The first Solar prototype provides an XML-based language to allow an application to describe its subscription tree. The leaves are simple name queries while all other operators are defined with a Java classname and the arguments that are necessary to initialize the instance. The Star looks in the name space to find matching sources or existing operators installed by other applications. For all other operators, the Star deploys a new instance on a randomly chosen Planet from the list of active Planets.

When asked to deploy an operator, the Planet loads the operator's Java class from the local CLASSPATH or remote code server and initializes a new instance with parameters supplied in the XML subscription request. The Planet maintains one outbound event queue for each resident source or operator, and a dedicated thread takes events from this queue and sends them to Planets hosting the subscribers. We multiplex operator subscriptions onto inter-Planetary TCP/IP sockets, so that there are at most two one-way TCP/IP connections between any two Planets, regardless of the number of operators on or subscriptions between the two Planets. The Planet's Network Manager thread monitors the inbound sockets and fills an inbound event queue; a dispatcher thread removes events from this queue and enters a reference for the event into the incoming event queue for each destination operator. Each operator has a dedicated thread to invoke the operator's event handler as new events arrive.

Solar provides an open programming framework for operator developers. Developers can write a new operator (in Java) by inheriting from the appropriate base class and implementing a few abstract methods. When an operator needs to publish an event, it simply calls an inherited *publish(IEvent)* method; the hosting Planet will capture the event and send to all its subscribers. An operator's *handleEvent(IEvent)* is automatically invoked when the Planet receives an event destined to that operator.

In our second prototype, we are implementing attribute-based naming and replacing the XML-based language with a more general composition language that is easier to use for selecting publishers and constructing the event-flow tree.

4.3 Future work

There are several research directions we plan to explore. Since Solar builds the operator graph incrementally and event publishing rates by the sources are generally unpredictable, we need a dynamic deployment algorithm to distribute (and redistribute) operators across Planets to balance load and minimize network traffic. Solar needs to support some kind of flow-control policies in the operator graph, without violating application semantics. For example, a fast publisher may want to disconnect slow subscribers, or slow down to wait until subscribers to catch up, or drop the events based on policies provided either by subscribers or the publisher itself. Solar also needs a general garbage-collection mechanism to delete operators with no subscribers.

To determine the value of the operator-graph abstraction and programming model, and the performance of the Solar system, we are developing and deploying several real-world context-sensitive mobile applications. We installed an IR-based location system to supply location context to our Solar system and its applications.[1] We plan to add more information sources to enrich the context space and to explore the performance and flexibility of the operator-graph abstraction.

5 Related work

Many have studied context-aware applications and their supporting systems. In Xerox Parc's distributed architecture each user's "agent" collects context (location) about that user, and decides to whom the context can be delivered based on that user's policy [16, 17].

A few projects specifically address the flexibility and scalability of context aggregation and dissemination. Like Solar, the Context Toolkit is a distributed architecture supporting context fusion and delivery [5]. It uses a *widget*

[1] See http://www.cs.dartmouth.edu/~solar/ for more information.

to wrap a sensor, through which the sensor can be queried about its state or activated. Applications can subscribe to pre-defined aggregators that compute commonly used context. Solar allows applications to dynamically insert operators into the system and compose refined context that can be shared by other applications. The Context Toolkit allows applications to supply filters for their subscriptions, while Solar introduces general filter operators to maintain a simple abstraction. IBM Research's context service, Owl, addresses similar issues such as scalability, extensibility, and privacy [6], but the paper provides no details.

Targeted for distributed sensor networks, Michahelles et al. propose to use context-aware packets to detect desired context [13]. These smart packets contain a retrieval plan that indicates what context sensors to visit to get results. The plan can be updated at run time according the results from certain sensors. The packets may also contain a *context hypothesis*, which can be evaluated at compute-empowered nodes, that derive higher-level context information based on the retrieved raw sensor data. At this point, it is unclear whether these smart packets could be used to deliver notifications about context changes.

Given the type of desired data, some systems automatically construct a data-flow path from sources to requesting applications, by selecting and chaining appropriate components from a system repository [10, 9]. CANS can further replace or rearrange the components to adapt to changes in resource usage [8]. To apply this approach to support context-aware applications, the system manager must foresee the necessary event transformations and install them in the component repository. These systems offer no specific support for applications to provide custom operators. Active Names, on the other hand, allow clients to supply a chain of generic components through which the data from a service must pass [18]. Also, Active Streams support event-oriented inter-process communication, and allow application-supplied *streamlets* to be dynamically inserted into the data path [7].

All of these approaches encourage the re-use of standard components to construct custom event flows. None, to our knowledge, specifically encourage the dynamic and transparent re-use of event streams across applications and users. Solar's re-use of operator instances, and their event streams, avoids redundant computation and data transmission, and improves scalability.

A non-procedural language, iQL, can specify the logic for composing pervasive data [4]. The model supports both requested and triggered evaluation. For one composer, iQL allows the inputs to be continually rebound to appropriate data sources as the environment changes. The language iQL complements Solar in two ways: iQL can be the programming language for individual operators, or iQL can be the high-level subscription language the compiler can decompose into a data-flow tree description used by Solar.

6 Summary

To support context-aware pervasive-computing applications, we propose an open platform, the "Solar" system, which employs a graph-based abstraction for context aggregation and dissemination. The abstraction models the contextual information sources as event publishers. The events flow through a graph of event-processing operators and become customized context for individual applications. This graph-based structure is motivated by the observation that context-aware applications have diverse needs, requiring application-specific production of context information from source data. On the other hand, applications do not have *unique* needs, so we expect there is substantial opportunity to share some of the processing between applications or users. Using a specification language, Solar allows applications to flexibly select contextual sources and construct their own event-flow tree. Solar then interconnects these trees to form a graph through reuse of named event streams.

We present the general model of Solar, discuss the details of the operator graph abstraction, and describe Solar's system architecture. We are using our Solar prototype to develop some context-aware applications [3, 12], and to support context-aware authorization [11]. We report some early experimental results in [3] and describe Solar's access-control model in [14]. We plan extensive additional research.

References

[1] Guruduth Banavar, Marc Kaplan, Kelly Shaw, Robert E. Strom, Daniel C. Sturman, and Wei Tao.

Information flow based event distribution middleware. In *ICDCS 1999*, Austin, Texas. IEEE Computer Society Press.

[2] Guanling Chen and David Kotz. Context aggregation and dissemination in ubiquitous computing systems. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society Press, June 2002.

[3] Guanling Chen and David Kotz. Solar: A pervasive computing infrastructure for context-aware mobile applications. Technical Report TR2002-421, Dept. of Computer Science, Dartmouth College, February 2002.

[4] Norman H. Cohen, Hui Lei, Paul Castro, John S. Davis II, and Apratim Purakayastha. Composing pervasive data using iQL. In *WMCSA 2002*, Callicoon, New York.

[5] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[6] Maria R. Ebling, Guerney D. H. Hunt, and Hui Lei. Issues for context services for pervasive computing. In *Workshop on Middleware for Mobile Computing 2001*, Heidelberg, Germany.

[7] Greg Eisenhauer, Fabian E. Bustamante, and Karsten Schwan. A middleware toolkit for client-initiated service specialization. *Operating Systems Review*, 35(2):7–20, April 2001.

[8] Xiaodong Fu, Weisong Shi, Anatoly Akkerman, and Vijay Karamcheti. CANS: Composable, adaptive network services infrastructure. In *USITS 2001*, San Francisco, California. USENIX.

[9] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2&3), 2001.

[10] Emre Kiciman and Armando Fox. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In *HUC 2000*, pages 211–226, Bristol, UK. Springer-Verlag.

[11] Chris Masone. Role Definition Language (RDL): A language to describe context-aware roles. Technical Report TR2001-426, Dept. of Computer Science, Dartmouth College, May 2002. Senior Honors Thesis.

[12] Arun Mathias. SmartReminder: A case study on context-sensitive applications. Technical Report TR2001-392, Dept. of Computer Science, Dartmouth College, June 2001. Senior Honors Thesis.

[13] Florian Michahelles, Michael Samulowitz, and Bernt Schiele. Detecting context in distributed sensor networks by using smart context-aware packets. In *ARCS 2002*, Karlsruhe, Germany. Springer-Verlag.

[14] Kazuhiro Minami and David Kotz. Controlling access to pervasive information in the “Solar” system. Technical Report TR2002-422, Dept. of Computer Science, Dartmouth College, February 2002.

[15] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *WMCSA 1994*, pages 85–90, Santa Cruz, California. IEEE Computer Society Press.

[16] William Noah Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, May 1995.

[17] Mike Spreitzer and Marvin Theimer. Providing location information in a ubiquitous computing environment. In *SOSP 1993*, pages 270–283, Asheville, NC. ACM Press.

[18] Amin Vahdat, Michael Dahlin, Thomas Anderson, and Amit Aggarwal. Active Names: Flexible location and transport of wide-area resources. In *USITS 1999*, Boulder, Colorado. USENIX.

[19] Abram White. Performance and interoperability in Solar. Technical Report TR2001-427, Dept. of Computer Science, Dartmouth College, June 2002. Senior Honors Thesis.